Informatika 1 – 2011

3. előadás – Sage alap függvényei

Steiner Mátyás

2011. szeptember 21.

Tartalom

Gyűjteményeken általánosan értelmezett függvények

	tuple	set	list	dict
új elem	-	`add`	`insert`, `append`	-
elem törlése	-	`remove`	`remove`	-
törlés mind	-	`clear`	-	`clear`
elem kivétele	-	`pop`	`pop`	`pop`
elem indexe	`index`	-	`index`	-
hányszor szerepel	`count`	-	`count`	-
másolás	-	`copy`	-	`copy`

Reference

- `add(<item>)` – új elem hozzáadása
- `insert(<index>,<item>)` – új elem beillesztése
- `append(<item>)` – új elem hozzáfűzése
- `remove(<item>)` – elem törlése
- `clear()` – gyűjtemény kiürítése
- `S:pop(), L: pop([<index>]), D: pop(<key>[, dv])` – egy elem visszaadása és törlése a gyüjteményből
- `index(<item>[, start [, stop]])` – elem indexe ha tartalmazza
- `count(<item>)` – megszámolja és visszatér hogy hányszor tartalmazza a gyűjtemény az adott elemet
- `copy()` – gyűjtemény lemásolása
- `len(<coll>)` – gyűjtemény hossza

Halmazokon értelmezett függvények - 1

Emlékeztető:

```
set = set([pi, 'abc', 35, pi])
```

Halmaz függvényei:

- `update(<coll>[, <coll>]*)`
  – union csak az eredmény bekerül a halmazba `(|=)`
- `union(<coll>[, <coll>]*)`
  – egyesíti a halmazokat, az eredménnyel visszatér
- `intersection(<coll>[, <coll>]*)`, `intersection_update(<coll>[, <coll>]*)`
  – visszaadja a közös részt ill. halmaz felülírása az eredménnyel `(&, &=)`

Halmazokon értelmezett függvények – 2

- `difference(<coll>[, <coll>]*)`, `difference_update(<coll>[, <coll>]*)`
  – különbség visszaadása ill. halmaz felülírása az eredménnyel (-, -=)
- `discard(<item>)`, `remove(<item>)`
  – elem törlése a halmazból, remove esetén hiba ha nem létezik
- `isdisjoint(coll)`
  – igaz ha nincs közös eleműek
- `issubset(<coll>)`
  – részhalmaza-e a paraméter `(<, <=)`
- `issuperset(<coll>)`
  – a halmaz részhalmaza-e a paraméternek `(>, >=)`

Lista függvényei

Emlékeztetőül: `list = ['a', 'orange', 1, 1.56]`

- `sort(cmp=None, reverse=False)`
  – rendezi a lista elemeit
- `extend(<coll>)`
  – hozzáadja az paraméterként átadott gyűjtemény elemeit a listához (+)
- `reverse()`
  – megfordítja a lista elemeinek sorrendjét
- * operátor – lemásolja és összefűzi újra a lista elemeit

Szótár függvényei – 1

Emlékeztetőül:

```
dict = {'a':4, 'orange':5, 1:5, 1.56:5.67}
```

- `get(<key>[, <dv>])`
  – a kulcshoz tartozó értékkel tér vissza ha létezik egyébként None ill. dv ha megadtuk
- `has_key(<key>)`
  – igaz ha a szótár eleme a kulcs
- `items()`
  – a szótár elemeit adja vissza egy listában (kulcs, érték) tuple formájában
- `iteritems()`
  – szótár elemein lépkedő iterátorral tér vissza
- `iterkeys()`
  – szótár kulcsain lépkedő iterátorral tér vissza

Szótár függvényei – 2

- `iteritems()`
  – szótár értékein lépkedő iterátorral tér vissza
- `popitem()`
  – szótár egy elemét eltávolítja és visszaadja (¡kulcs¿, ¡érték) tuple formájában
- `keys()`
  – a kulcsokat adja vissza egy listában
- `update(<coll>)`
  – dict vagy tuple lista elemeit illeszti be a szótárba
- `setdefault(<key>[, value])`
  – beilleszti az elemet és visszatért az értékkel
- `values()`
  – visszatér az értékekből képzett listával

Függvények számításokhoz – 1

- `var(<string>)`
  – szimbolikus számításokhoz hozhatunk létre változókat
- `subs(<args>)`
  – kifejezés függvénye, amivel behelyettesíthetjük a változókat a paraméterben felsorolt értékekkel és visszakapjuk a kifejezés numerikus eredményét
- `float(<kif|num|string>)`
  – lebegőpontos számmá próbálja alakítani a megadott paramétert (lsd. RR, RDF)
- `n()`
  – kifejezések függvénye, a kiértékelt kifejezést adja vissza

Függvények számításokhoz – 2

- `exp(3)`
  $= e^3$
- `pow(x,z)`
  $= x^z$
- `log(exp(3))` – természetes alapú log
  $= 3$
- `sqrt(16)` – négyzetgyök
  $= 4$
- `sin(pi), cos(pi), tan(pi)` – szögfüggvények
  `arcsin(), ...` – inverzeik
- `floor(4.5)` – alsó egészrész
  $= 4$
- `ceil(4.6)` – felső egészrész
  $= 5$

Egyenlőség vizsgálata lebegőpontos számok esetében

Fontos megjegyezni, hogy a lebegőpontos számok mindig közelítő értékek, így két lebegőpontos szám, ami pl. számítás eredménye, valószínűleg nem egyezik meg egymással!
$sqrt(4.0)^2 == 4.0$ `--> False`

Megoldás: $abs(sqrt(4.0)^2 - 4.0) < 10^{-10}$

Egyenlőség vizsgálata lebegőpontos számok esetében

Fontos megjegyezni, hogy a lebegőpontos számok mindig közelítő értékek, így két lebegőpontos szám, ami pl. számítás eredménye, valószínűleg nem egyezik meg egymással!
$sqrt(4.0)^2 == 4.0$ `--> False`

Megoldás: $abs(sqrt(4.0)^2 - 4.0) < 10^{-10}$

Kifejezéseken használt függvények

Az algebrai kifejezéseket összeggé alakíthatjuk az `expand()` függvénnyel vagy `.expand()` metódussal, ill. szorzattá a `factor()`-ral:

- `sage: expand((a+b)^2)`
  $a^2 + 2 * a * b + b^2$
- `sage: (x^2-1).factor()`
  $(x - 1) * (x + 1)$

Kifejezéseken használt függvények

A kifejezéseket a `simplify()` vagy a `full_simplify()` metódussal tudjuk egyszerűsíteni.

```
t = p/p**x
t.simplify()
p**(-x + 1)
```

```
t = (x + 1)**3-x**3-1
t.full_simplify()
3*x**2 + 3*x
```

Egyenletek megoldása

- `solve()` – megpróbálja szimbolikus átalakításokkal meghatározni az egyenlet gyökét
- `roots()` – mint a solve, csak az eredményt numerikusan és a gyök multiplicitását adja vissza

```
(x + 1/x == 4).solve(x)
[x == -sqrt(3) + 2, x == sqrt(3) + 2]
(x**2 + 2*x + 1 == 0).roots(x)
[(-1, 2)]
```

Egyenletek megoldása – 2

Az előbbi metódusok néha nem tudnak explicit alakban megoldani egy egyenletet, vagy nem ad meg minden gyököt, vagy hamis gyököket talál. Ekkor a `find_root()` segíthet egy numerikus megoldást találni a megadott intervallumon (legyen pl: $0 < y < \pi/2$):

- ```
  sage:  y = var('y')
  sage:  find_root(cos(y) == sin(y), 0, pi/2)
  0.78539816339744839
  ```

Az algoritmus mindig csak egy gyököt talál meg és az is egy közelítő érték.
  ```

Max, min keresése

Maximumot vagy minimumot is kereshetünk numerikusan. Ez is feltételezi, hogy a bemenet egy egyváltozós folytonos függvény, és egy lokális maximumot keres meg közelítőleg.

- `find_maximum_on_interval(start, end)`
- `find_minimum_on_interval(start, end)`

Határérték, derivált

Kiszámíthatjuk egy kifejezés határértékét egy pontban, vagy a deriváltját.

- határérték
  ```
  ((2**x - 1)/sin(x)).limit(x = 0)
  ```
  $-\log(2)$
- derivált számítása `(2**x - 1).derivative(x)`
  ```
  2**x*log(2)
  ```
- ellenőrizhetjük:
  ```
  (2**x - 1).derivative(x).subs(x=0)
  log(2)
  ```